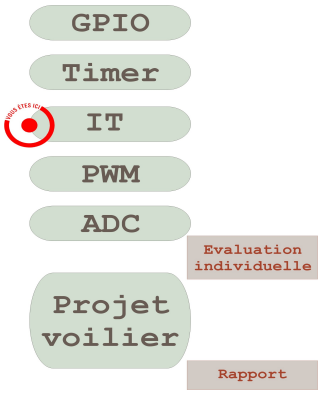
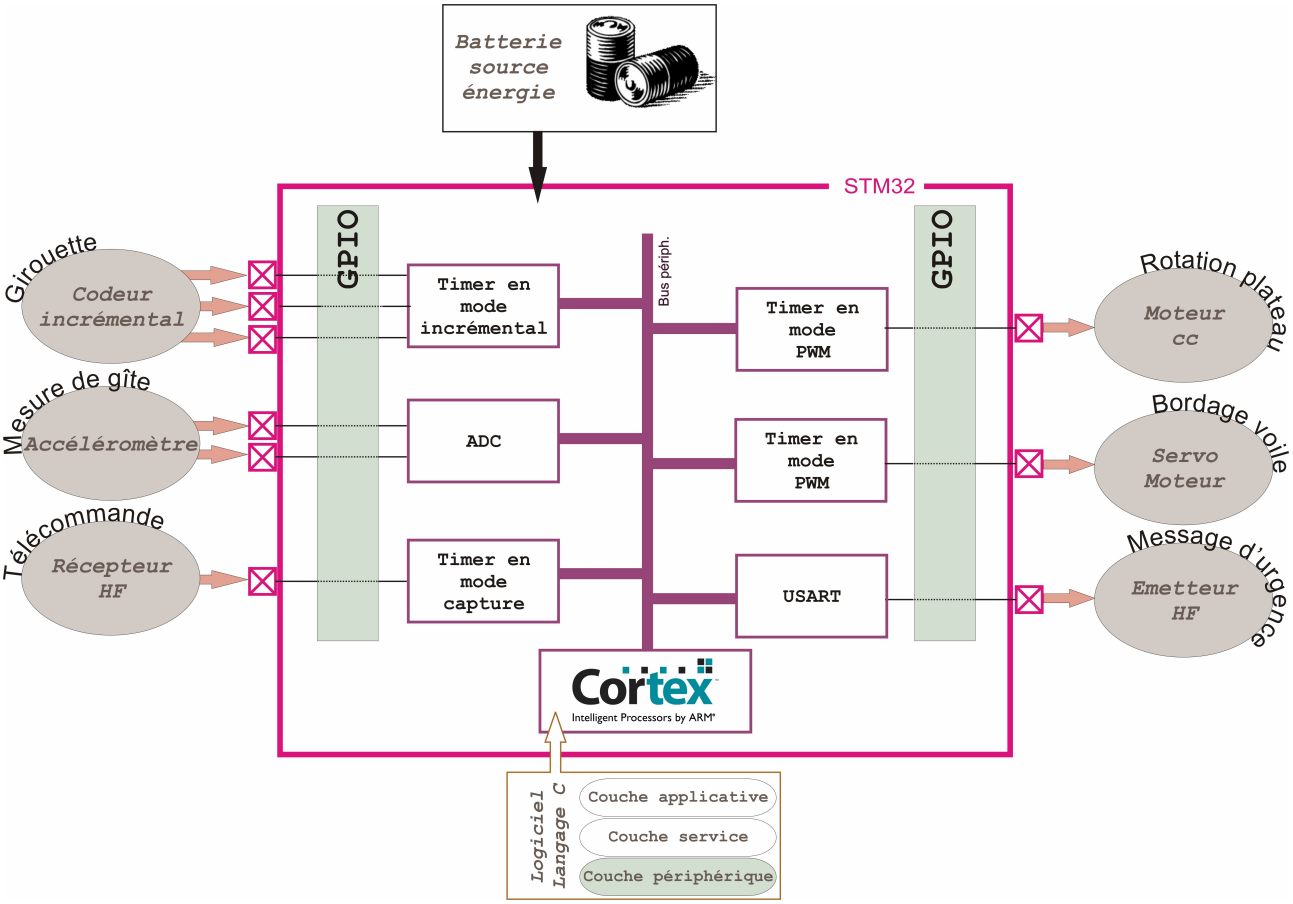


TP *sur les interruptions*

1- Les interruptions dans le système *Bordage de voile automatique d'un voilier*
2- Le système d'interruption en général
3- Travail de l'étudiant – fonctionnement des IT du STM32

1 Les interruptions dans le système *Bordage de voile automatique d'un voilier*



Le système d'interruptions va nous être utile dans le cadre du projet :

- Pour organiser le déroulement en temps réel du programme applicatif : l'interruption permet de dérouter automatiquement le programme principal vers un "handler" (programme d'interruption) au moment où un timer déborde (association Timer-IT).
- Pour déterminer automatiquement la durée de l'impulsion du récepteur HF (association Timer-IT)
- Pour lire automatiquement les valeurs acquises par l'ADC (association ADC-IT)
- Pour gérer la référence absolue de la girouette (association broche-IT)

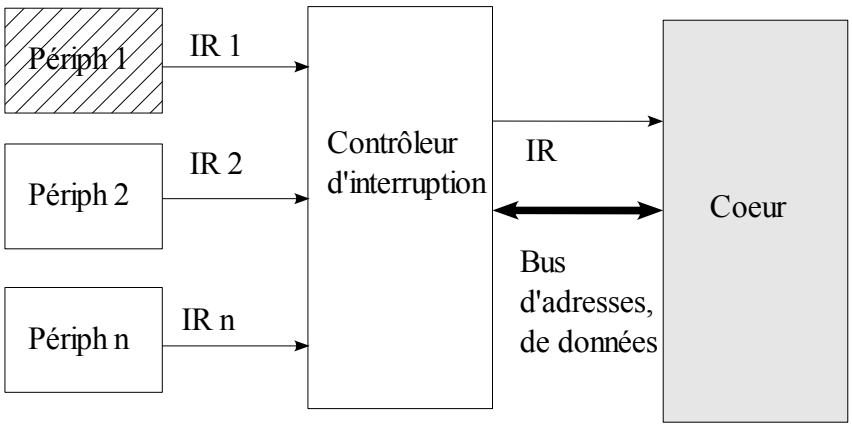
2 Le système d'interruption en général

Le *système d'interruption* est un des éléments essentiels d'un microcontrôleur. Il permet de décharger le processeur (minimiser son travail de traitement logiciel), pour laisser des actions se faire de manière matérielle. Typiquement, surveiller l'état d'un périphérique est bien inutile. Mieux vaut laisser au périphérique le soin de signaler lui-même son état. C'est donc bien le système d'interruptions qui va permettre de mettre en place ce type de dialogue coeur – périphérique, qui va donner en quelques sortes de “l'autonomie” au périphérique.

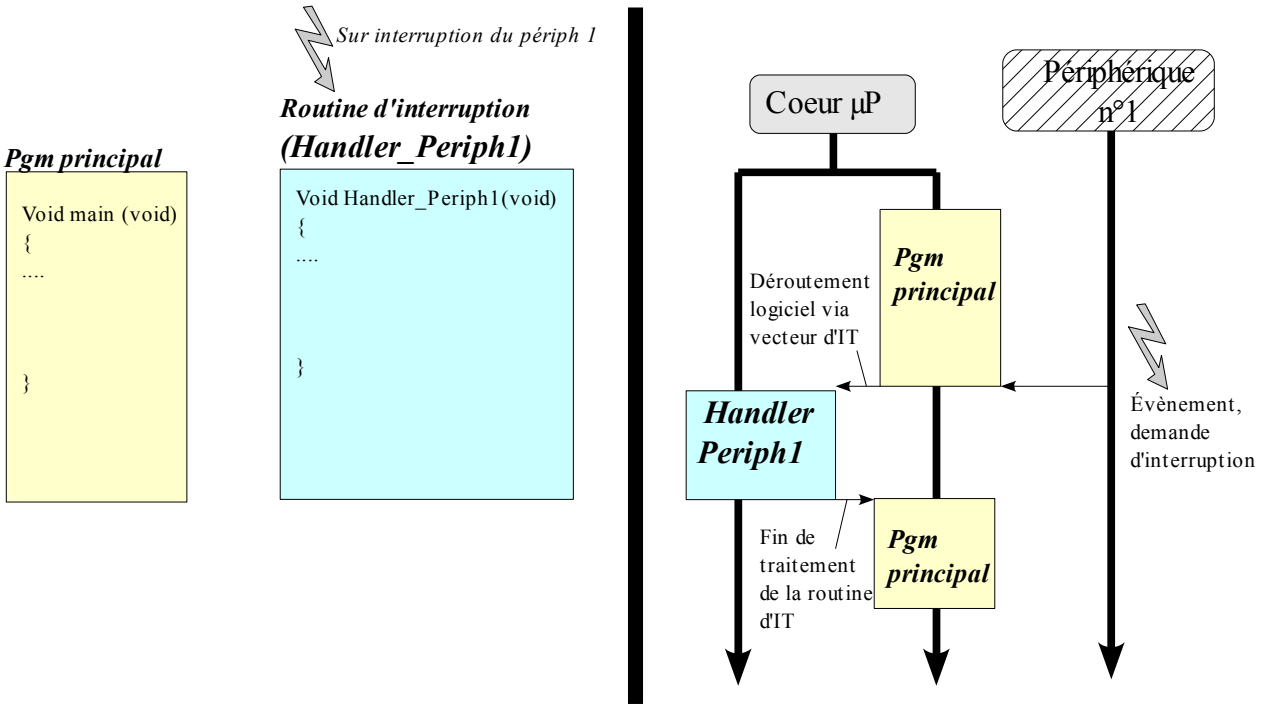
En principe, un périphérique donné possède un fil de sortie que l'on peut appeler *Interrupt Flag*, ou encore *Interrupt Request*. Le périphérique est relié au coeur, via un *gestionnaire d'interruptions*. Ce dernier est capable d'interrompre le coeur et de lui fournir une adresse où est situé le *programme d'interruption* à traiter (le *Handler*). Cette adresse est connue sous le nom de *vecteur d'interruption*.

Finalement, si chaque périphérique possède son propre vecteur d'interruption, alors il est très simple de dérouter le coeur de son activité pour l'aiguiller directement sur le programme correspondant.
Parfois, pour des raison de simplicité de fabrication, plusieurs périphériques, possèdent le même vecteur d'interruption. Dans ce cas, le programmeur doit prévoir un test sur les *Interrupt Flag*, afin de savoir qui est à l'origine de l'interruption et donc de lancer le traitement approprié.

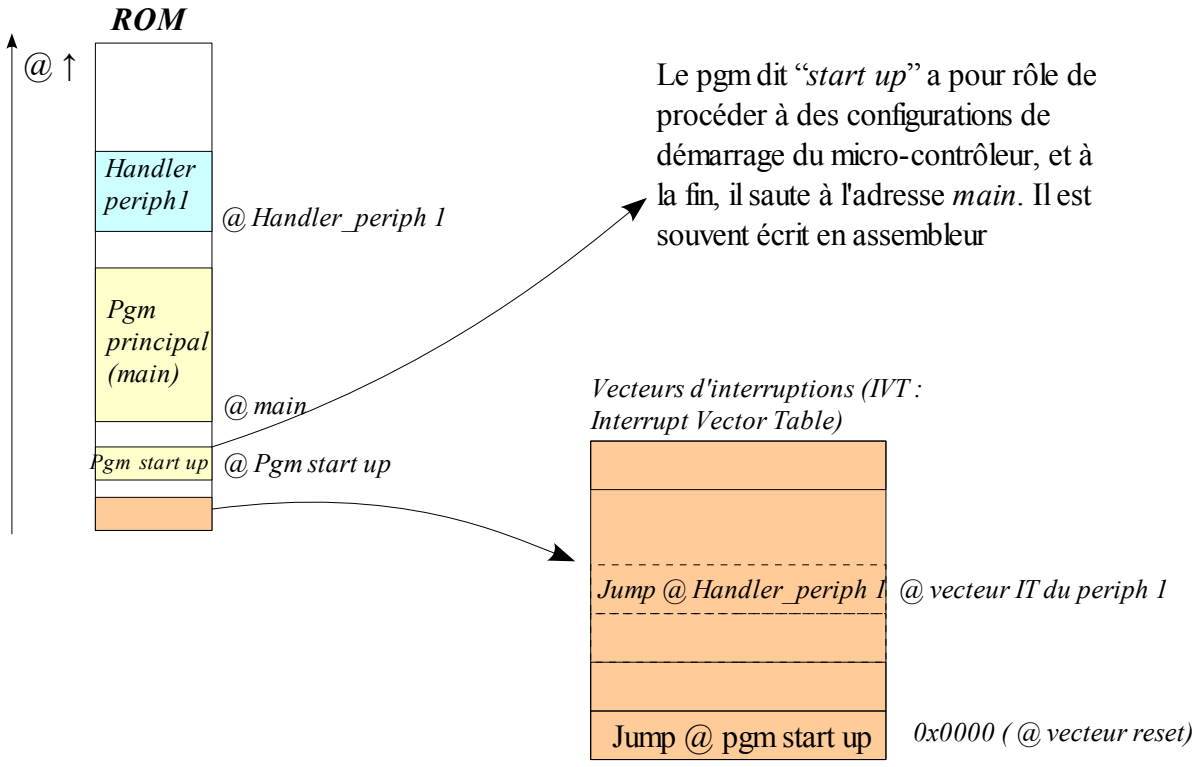
2.1 Schéma très simplifié de la structure matérielle



2.2 Structure logicielle et évolution dans le temps



2.3 Schéma représentant les espaces ROM



Précisons que le contrôleur d'interruption permet aussi de gérer les priorités d'interruptions (dans le cas où plusieurs sont demandées en même temps). Enfin, pour qu'un périphérique puisse émettre une interruption il faut d'une part que la condition de réalisation soit rencontrée (débordement d'un timer par exemple), mais il faut aussi que la demande soit autorisée. Très souvent, au sein d'un périphérique, on trouve un bit qu'on peut appeler *IT_Periph_IE*, où *IE* signifie *Interrupt Enable*. Signalons enfin, que dans de nombreux cas, un bit de commande global, *GIE* (*Global Interrupt Enable*), appartenant au coeur permet de bloquer ou pas l'ensemble des interruptions validées localement dans chaque périphérique.

3 Travail de l'étudiant – fonctionnement des IT du STM32

Le périphérique incontournable pour mettre en place une interruption est le ***NVIC*** (*Nested Vectored Interrupt Controller*). Il fait partie du coeur *Cortex-M3*, non des périphériques. C'est pourquoi les informations sont à prendre dans les deux documentations *PM0056 Programming manual* et *RM0008 Reference manual.*

Nous allons à titre d'exemple nous intéresser aux interruptions des Timers 2, 3 et 4.

Questions guide :

- *(cf. RM0008)* Quelles sont les 4 types d'interruptions possibles susceptibles d'être émises par un module timer (General Purpose Timer) ? Quel est celui qui nous intéresse pour générer une interruption sur débordement ?
- *(cf. RM0008)* Quel est le rôle des registres ***DIER*** et ***SR*** des Timers 2, 3 et 4 ?
- Une fois le programme main dérouté vers le programme d'interruption (*Handler*), comment savoir l'origine précise de l'interruption ? Que convient-il de faire immédiatement après l'entrée dans le *Handler* afin de ne pas repartir en interruption dès la sortie du *Handler* ?
- *(cf. RM0008)* Le ***NVIC*** permet de gérer un certain nombre de demandes d'interruptions (internes au coeur et externe). A chaque demande est associé un *vecteur d'interruption* (une adresse physique) ainsi qu'une *position* (un numéro). Combien de demandes d'interruptions le NVIC peut-il gérer ? Pour chacun des 3 Timers, préciser la *position* dans la table IVT correspondant à une demande d'interruption sur débordement du Timer.
- *(cf. PM0056)* Pour une ligne d'interruption, le NVIC affecte une priorité. Indiquer la plage de valeurs possibles pour la priorité, ainsi que la valeur la plus prioritaire.
- *(cf. PM0056)* Quels sont les registres qui permettent de régler la priorité d'une interruption.
- *(cf. PM0056)* Expliquer brièvement le rôle des groupes de registres ***ISER, ICER, ISPR, ICPR*** *et* ***IABR.***

Vous avez répondu à toutes ces questions ? Si oui, vous pouvez vous lancer dans la programmation d'une interruption.

4 Travail de l'étudiant – Keil

4.1 Mise en place de l'interruption de débordement du Timer 2

⌦ Créer un nouveau projet et configurer dans le *main(),* le Timer 2 pour qu'il produise une interruption toutes les 500ms.

⌦ Configurer l'interruption et activez la.

⌦ Ecrire la routine d'interruption dans le fichier main() pour faire clignoter une diode.

NB: Pour écrire la fonction d'interruption en C, chaque compilateur a son style ! En ce qui concerner le compilateur ARM et GCC, la syntaxe est :

```
void TIM2_IRQHandler (void)
```

Ce nom ne s'invente pas, il se trouve écrit dans le fichier startup **STM32F10x.s**

4.2 Nouvelle version d'un pilote

⌦ Ajouter au pilote timer une fonction permettant de configurer de l'interruption d'un Timer.

```
/**
* Configure la routine d'interruption d'un Timer
* @param Timer Pointeur vers le jeu de registres (de type TIM_TypeDef ) du
* timer considéré
* @param Priority Niveau de priorité de l'interruption
* @param IT_function Pointeur sur la fonction qui sera exécutée dans le routine
d'interruption
**/
void Timer_Active_IT( TIM_TypeDef *Timer,
                      u8 Priority,
                      void (*IT_function (void)  );
```

⌦ Testez votre fonction.